Global Predicate Detection

Slides are based on the book chapter from Distributed Computing: Principles, Algorithms and Systems (Chapter 11) by Kshemkalyani and Singhal

Pradipta De
pradipta.de@sunykorea.ac.kr
Global Predicate?

• Applications in process control, distributed debugging, sensor networks

• Predicate examples:
  – (Pressure-A > 240 KPa) and (Temp > 300 C)
  – B = x_i + y_j + z_k < -125
  – Stop the program when (x_1 in P_1) and (x_2 in P_2)

• Difference with Global snapshot detection
  – Snapshot Detection: gives one of the possible states that the system could have existed during execution
Types of Predicates

• Stable
  – A global predicate \( B \) is stable if once it becomes true, it stays true.
  – A predicate \( \varphi \) at a cut \( C \) is stable if the following holds
    • \( (C \models \varphi) \Rightarrow (\forall C' \mid C \subseteq C', C' \models \varphi) \)

• Examples:
  – Deadlock Detection
  – Termination detection
  – Distributed debugging

• Observation: Each stable property can be characterized by,
  – Local condition: process state
  – Global condition: channel state
Types of Predicates

• Unstable
  – A predicate that may hold only intermittently
  – Challenging to detect:
    • Different executions may pass through different global states
    • Predicate may be true in some execution and false in others
    • No global clock makes it impossible to get an instantaneous snapshot
    • Intermittent monitoring may miss a state
Detecting Stable Predicate

Naïve approach: Repeated global snapshot!! Impractical
O\(n^2\) control messages without blocking processes

Two-phase observation of global states: Observe (potentially inconsistent) global states ... all local and global variables are observed in each state

Stable property is true if,
No change in the variables in the 2 observed states

One Implementation: O(n) control messages
1. Each process records its state variables and sends to a central process ;; central process informs all other processes to send its state after receiving the message
Modalities on Predicates

- Possibly($\varphi$): There exists a consistent observation of the execution such that $\varphi$ holds in a global state of the observation.

- Definitely($\varphi$): For every consistent observation of the execution, there exists a global state of it in which predicate $\varphi$ holds.
Illustration of modality

Definitely( a+b = 10 )
Possibly( a+b = 5 )

Complexity: O(m^n) where m events at each of n processes

Global Predicate Detection is an NP-Complete problem
Understanding the Key Problems

• Lack of shared clock
• Assume a predicate $B = S_1 \cap S_2$, where $S1$ and $S2$ are predicates local on processes $P1$ and $P2$
  – $B$ is true iff there is an instant of time when $S1$ and $S2$ are true
  – Impossible to determine since clocks are not synchronized

• Use happened-before relation to replace time
  – $B$ is true if $S1$ and $S2$ are true and the events corresponding to $S1$ and $S2$ are concurrent
Understanding the Key Problems

- Lack of shared memory $\implies$ inherent communication complexity
  - If evaluation of a function requires every change in variable values from all processes, then impractical in practice
  - Use the notion of monotonicity to send only one value per external event
    - If detecting $(x_1 > x_2)$ and $x_1 = \{2, 9, 4, 7\}$, then sufficient to communicate 9.
  - A variable value is monotone if it takes its values from a totally ordered set
Understanding the Key Problems

• Combinatorial Explosion: With n processes, and m states, total number of possible global states is $m^n$

• Global predicate detection is a decision problem which can be mapped to boolean satisfiability problem
  – NP complete problem

• Restrict the class of predicates for efficient detection
  – Stable predicates are a sub-class of observer independent predicates
Relational Predicate Detection: Centralized Algorithm

- Assume state lattice is available
- Global state $GS = \{s_{1}^{k_{1}}, s_{2}^{k_{2}}, \ldots, s_{n}^{k_{n}}\}$ is written as $GS^{k_{1}, k_{2}, \ldots, k_{n}}$
  - $s_{1}^{k_{1}}$ denotes an event state in the process $p_{1}$, while $GS$ is a node in the state lattice
- Level of a global state in lattice is calculated as,
  - $\langle s_{i}^{k_{i}} (\forall i) \rangle = \sum_{i=1}^{n} k_{i}$
- Possibly($\varphi$) : exhaustive search of state lattice for one node where the predicate is satisfied
  - Finding the match at lower levels gives faster match
- Definitely($\varphi$) : find a set of states satisfying $\varphi$, not necessarily at the same level, such that every path through the lattice goes through one of these states.
At each level, all the states are available

If any one state satisfies predicate, then Possible(\(\varphi\)) holds

If for all paths, there is a node where predicate holds true, then Definitely(\(\varphi\)) holds.
Relational Predicates: Centralized Algo

(variables)
set of global states $\text{Reach}_\phi, \text{Reach\_Next}_\phi \leftarrow \{GC^{0,0,...0}\}$
int $lvl \leftarrow 0$

(1) **Possibly**($\phi$)
(1a) while (no state in $\text{Reach}_\phi$ satisfies $\phi$) do
(1b) if ($\text{Reach}_\phi = \{\text{final state}\}$) then return false;
(1c) $lvl \leftarrow lvl + 1$;
(1d) $\text{Reach}_\phi \leftarrow \{\text{states at level } lvl\}$;
(1e) return true.

(2) **Definitely**($\phi$)
(2a) remove from $\text{Reach}_\phi$ those states that satisfy $\phi$
(2b) $lvl \leftarrow lvl + 1$;
(2c) while ($\text{Reach}_\phi \neq \emptyset$) do
(2d) $\text{Reach\_Next}_\phi \leftarrow \{\text{states of level } lvl \text{ reachable from a state in } \text{Reach}_\phi\}$;
(2e) remove from $\text{Reach\_Next}_\phi$ all the states satisfying $\phi$;
(2f) if $\text{Reach\_Next}_\phi = \{\text{final state}\}$ then return false;
(2g) $lvl \leftarrow lvl + 1$;
(2h) $\text{Reach}_\phi \leftarrow \text{Reach\_Next}_\phi$;
(2i) return true.
Implementation

• States are not available a priori $\Rightarrow$ global states must be constructed on-line based on local states sent by each process

• Each process $P_i$ sends a local trace of its local states $s^{ki}_i$, with vector timestamps, to central process $P_0$
  
  – States are stored in Queues, one for each process

Fig 11.4: Queues $Q_1 \ldots Q_n$ for each of the $n$ processes
# Implementation

- **How long are states held in a queue?**
  - A state can be discarded when it is sure that it will not be requested anymore to construct the global state.

- **Observations on Vector Clock timestamp**

The earliest global state $G_{s_{min}}^{k_1,k_2,...,k_n}$ containing $s_{ij}^{k_i}$ is identified as follows. The $j^{th}$ component of $VC(s_{ij}^{k_i})$ is the local value of $P_j$ in its local snapshot state $s_{ij}^{k_j}$.

$$(\forall j) \ VC(s_{ij}^{k_j})[j] = VC(s_{ij}^{k_i})[j]$$ (1)

Thus, the lowest level of the state lattice, in which local state $s_{ij}^{k_i}$ ($k^{th}$ local state of $P_i$) participates, is the sum of the components of $VC(s_{ij}^{k_i})$.

The latest global state $G_{s_{max}}^{k_1,k_2,...,k_n}$ containing $s_{ij}^{k_i}$ is identified as follows. The $i^{th}$ component of $VC(s_{ij}^{k_j})$ should be the largest possible value but cannot exceed or equal $VC(s_{ij}^{k_i})[i]$ for consistency of $s_{ij}^{k_i}$ and $s_{ij}^{k_j}$. $VC(s_{ij}^{k_i})$ is identified as per Equation 2; note that the condition on $VC(s_{ij}^{k_j+1})[i]$ is applicable if $s_{ij}^{k_j}$ is not the last state at $P_j$.

$$(\forall j) \ VC(s_{ij}^{k_j})[i] < VC(s_{ij}^{k_i})[i] \leq VC(s_{ij}^{k_j+1})[i]$$ (2)

Hence, the highest level of the state lattice, in which local state $s_{ij}^{k_i}$ participates, is $\sum_{j=1}^{n} VC(s_{ij}^{k_j})[j]$. 
Conjunctive Predicates

- A predicate $\varphi$ is conjunctive iff $\varphi$ can be expressed as a conjunction of $\varphi_i$s, where $\varphi_i$ is a predicate local to process $P_i$

- Property of conjunctive predicates:
  - If $\varphi$ is false in any cut $C$, then there is at least one process $i$ such that the local state of $i$ will not form part of any other cut
  - $\Rightarrow$ advance the local state of process $i$ since $C[i]$ is a forbidden state
Examples

Possibly \((a=3 \cap b=2)\) ?

Definitely \((a=3 \cap b=7)\) ?
Conjunctive Predicate Detection

- **Interval-based approach**

  For two processes:

  - Definitely($\phi$) : $\min(X) < \max(Y) \land \min(Y) < \max(X)$
  - Possibly($\phi$) : $\max(X) < \min(Y) \lor \max(Y) < \min(X)$

  For multiple processes:

  - Definitely($\phi$) : $\bigwedge_{i,j \in \mathbb{N}}$ Definitely($\phi_i \land \phi_j$)
  - Possibly($\phi$) : $\bigwedge_{i,j \in \mathbb{N}}$ Possibly($\phi_i \land \phi_j$)

- **Global state based approach**

  - Examines the global states

\[\begin{align*}
\text{Definitely}(\phi) & : \min(X) < \max(Y) \land \min(Y) < \max(X) \\
\text{Possibly}(\phi) & : \max(X) < \min(Y) \lor \max(Y) < \min(X)
\end{align*}\]
Possibly(\varphi): Global State-based Algo

• Detecting Possibly(\varphi) is equivalent to detecting a consistent state where local state at each process \(P_i\) satisfies \(\varphi_i\)
  – \(s_i\) and \(s_j\) are mutually concurrent
    (mutually concurrent) \(\forall i, \forall j, s_i \not\equiv s_j \land s_j \not\equiv s_i\)
Possibly(φ): Global State-based Algo

```
integer: GS[1... n, 1... n]; //ith row tracks vector time of Pi
boolean: Valid[1... n]; //Valid[j] = 0 implies Pj state GS[j, .] to be advanced
queue of array of integer: Q1, Q2, ..., Qn ← ⊥; //Qi stores timestamp info from Pi
(1) while (∃j | Valid[j] = 0) do //Pj’s state GS[j, .] is not consistent with others
(2) if (Qj = ⊥ and Pj has terminated) then
(3) return(0);
(4) else
(5) await Qj becomes non-empty;
(6) GS[j, 1... n] ← head(Qj); //Consider next state of Pj for consistency
(7) dequeue(head(Qj));
(8) Valid[j] ← 1;
(9) for k = 1 to n do //Check Pj’s state w.r.t. Pk’s state (for every Pk)
(10) if k ≠ j and Valid[k] = 1 then
(11) if GS[j, j] ≤ GS[k, j] then //Pj’s state is inconsistent with Pk’s state
(12) Valid[j] ← 0; //next state of Pj needs to be considered
(13) else if GS[k, k] ≤ GS[j, k] then //Pk’s state inconsistent with Pj’s state
(14) Valid[k] ← 0; //next state of Pk needs to be considered
(15) return(1).
```
Distributed Algo for Possibly

- Uses a token passing mechanism
  - For global-state based technique
  - For interval based technique